

Programmierung Übung zu TCP mit python

Auf dieser [Seite](#) gibt es eine ausführliche Beschreibung der TCP-API von python.

Aufgaben

1. Programmieren Sie eine einfache Client-Client-Anwendung bei der sich die jeweiligen Clients Nachricht per TCP zusenden.
2. Erweitern Sie den Server zu einem Multi-Connection-Server, der mehr als eine Verbindung bedienen kann.
3. Geben Sie eingehende Nachrichten an alle anderen Verbindungen aus.

Hinweise / Tipps

Folgender Code kann als Grundlage für einen sogenannten ECHO-Server dienen, der alle Daten an den Sender zurück schickt, die er empfangen hat.

Der Code wird [hier](#) ausführlich erklärt.

ECHO-Server

Der Server binden sich auf den Socket (HOST, PORT) und lauscht (conn.recv) dort auf einkommende Daten. Die empfanenen Daten sendet er an den Client per conn.sendall direkt zurück. Zur Kontrolle gibt er per print die empfangenen Daten in der Shell aus.

[echo_server.py](#)

```
#!/usr/bin/env python3

import socket

HOST = '127.0.0.1' # Hier sollte eine "echte" IP eingetragen werden,
                  # ansonsten kann der Server nur von dem PC selbst erreicht werden
PORT = 65432      # Port auf dem gelauscht (listen) wird (nicht
                  # privilegierte Ports > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(0) # Auf der Real Python-Seite fehlt der
               # Parameter 0, so dass es zu Fehlermeldungen kommt
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
```

```
data = conn.recv(1024)
if not data:
    break
conn.sendall(data)
print('Client says:', repr(data))
```

ECHO-Client

Der Client schickt genau eine Nachricht an den Server und gibt per print das Echo (Antwort des Servers) in der Shell aus.

[echo_client.py](#)

```
#!/usr/bin/env python3

import socket

HOST = '127.0.0.1' # Hostname oder IP des Echo-Servers eintragen
PORT = 65432      # Port des Servers eintragen

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)

print('Received', repr(data))
```

Interaktiver Echo-Client

[echo_client_interactive.py](#)

```
#!/usr/bin/env python3

import socket

HOST = '127.0.0.1' # Hostname oder IP des Echo-Servers eintragen
# PORT = 65432      # Port des Servers eintragen
PORT = 50000

host = input('Server-IP [' + HOST + ']: ')
port = input('Server-Port [' + str(PORT) + ']: ')

if host != '':
    HOST = host
if port != '':
```

```
PORT = int(port, 10)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    written = ''
    while True:
        written = input('Your message (type "exit" to exit): ')
        if written == 'exit':
            break;
        s.sendall(bytearray(written, 'UTF-8'))
        data = s.recv(1024)
        print('Received ', repr(data))

print('Client closed')
```

Multi-Connection-Server

Quelle: Das Beispiel stammt ursprünglich von dieser [Seite](#) und wurde leicht angepasst bzw. für python3 lauffähig gemacht. Als Client kann der ECHO-Client von oben verwendet werden. Jede Anfrage führt zu einer neuen Verbindung.

Diese Verbindungen können mittels:

```
netstat -an | grep 50000
```

angezeigt werden. Hier ist 50000 der LISTEN-Port des Servers. Ggf. muss localhost durch eine erreichbare IP-Adresse des betreffenden Server-PCs ausgetauscht werden.

[tcp_multi_server.py](#)

```
#!/usr/bin/env python3

import select, socket, sys, queue
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setblocking(0)
server.bind(('localhost', 50000))
server.listen(5)
inputs = [server]
outputs = []
message_queues = {}

while inputs:
    readable, writable, exceptional = select.select(
        inputs, outputs, inputs)
    for s in readable:
        if s is server:
            connection, client_address = s.accept()
            connection.setblocking(0)
```

```
        inputs.append(connection)
        message_queues[connection] = queue.Queue()
    else:
        data = s.recv(1024)
        if data:
            message_queues[s].put(data)
            if s not in outputs:
                outputs.append(s)
            else:
                if s in outputs:
                    outputs.remove(s)
                inputs.remove(s)
                s.close()
            del message_queues[s]

    for s in writable:
        try:
            next_msg = message_queues[s].get_nowait()
        except queue.Empty:
            outputs.remove(s)
        else:
            s.send(next_msg)

    for s in exceptional:
        inputs.remove(s)
        if s in outputs:
            outputs.remove(s)
        s.close()
        del message_queues[s]
```

Keystroke als Command Line Handler

ACHTUNG: Das folgende Beispiel funktioniert nur, wenn das Script direkt in der Shell ausgeführt wird. Hierzu muss zunächst die Datei ausführbar gemacht werden. Mit folgendem Befehl kann dies getan werden:

```
chmod +x keystroke_sample.py
```

Das folgende Beispiel wartet dauerhaft auf eine Eingabe und führt je nach Tastedruck Programmcode aus.

[keystroke_sample.py](#)

```
#!/usr/bin/env python3

import sys, termios, tty, os, time

def getch():
```

```
fd = sys.stdin.fileno()
old_settings = termios.tcgetattr(fd)
try:
    tty.setraw(sys.stdin.fileno())
    ch = sys.stdin.read(1)

finally:
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
return ch

while True:
    char = getch()

    if (char == "p"):
        print("Stop!")
        exit(0)

    if (char == "a"):
        print("Left pressed")
        time.sleep(1)
```

From:

<https://www.kopfload.de/> - **kopfload - Lad Dein Hirn auf!**

Permanent link:

https://www.kopfload.de/doku.php?id=lager:lok_netze:tcp_python

Last update: **2025/11/19 16:15**

